

Trading Order Matching Engine - Assignment

Objective

Build an order matching engine that processes buy and sell orders for different trade types, validates them according to business rules, and matches compatible orders while handling concurrent order submissions.

Problem Statement

You need to create a system that:

1. Accepts buy and sell orders from multiple traders for different trade types
 2. Validates orders against business rules
 3. Maintains separate order books for each trade type
 4. Matches compatible buy and sell orders based on price and timestamp (FIFO for same price)
 5. Executes trades when matches are found
 6. Handles concurrent order submissions safely
-

Functional Requirements

Order Properties

Each order must contain:

- Unique order ID
- Trader ID
- Trade type (EQUITY, FOREX, CRYPTO)
- Order type (BUY or SELL)
- Price
- Quantity
- Country code (2-letter ISO code)
- Timestamp

Validation Rules

1. **Country Code Validation:** Only allow orders from approved countries (US, UK, IN, SG, JP, DE, FR)
2. **Maximum Amount Validation:**

- **EQUITY:** Maximum order value (price × quantity) = 100,000
- **FOREX:** Maximum order value = 500,000
- **CRYPTO:** Maximum order value = 50,000

3. **Data Validation:** All fields must be valid (non-null, positive values, etc.)

Matching Logic

- Buy and sell orders match when:
 - Same trade type
 - Buy price ≥ Sell price
 - Both orders have available quantity
- Orders with the same price are matched based on timestamp (earlier orders first)
- Partial fills are allowed (order can match with multiple counter-orders)
- **Sell orders remain active** until their entire quantity is covered by one or more buy orders
- **Buy orders remain active** until their entire quantity is covered by one or more sell orders

Order Status

Orders should transition through states: PENDING → PARTIALLY_FILLED → FILLED (or REJECTED)

- An order moves to FILLED only when its entire quantity has been matched
 - Orders stay active in the order book until fully filled
-

Non-Functional Requirements

Concurrency

- System must handle multiple traders submitting orders simultaneously
- Order book must remain consistent under concurrent access
- Matching process must be thread-safe
- No race conditions when updating order quantities

Exception Handling

Handle and report appropriate exceptions for:

- Invalid country codes
- Amount limit violations
- Duplicate order IDs

- Invalid order data
- Concurrent modification conflicts
- Insufficient quantity for matching

Extensibility

Design should allow easy addition of:

- New trade types with different rules
 - New validation rules
 - Different matching strategies
 - New notification mechanisms
-

Deliverables

1. **Source Code:** Well-structured Java code demonstrating design patterns and best practices
 2. **Test Suite:** Comprehensive tests using JUnit/TestNG covering:
 - Single order validation scenarios
 - Successful matching scenarios
 - Concurrent order submission
 - Exception handling cases
 - Edge cases (partial fills, multiple matches, etc.)
 3. **README:** Brief documentation explaining:
 - How to run the application
 - Design patterns used and why
 - Concurrency approach
 - Any assumptions made
-

Evaluation Criteria

Design Patterns (25%)

- Appropriate use of design patterns for extensibility
- Clean separation of concerns
- SOLID principles adherence

Low-Level Design (25%)

- Class structure and relationships
- Interface design
- Code organization and modularity

Concurrency (25%)

- Thread-safety of shared data structures
- Proper synchronization mechanisms
- Handling concurrent modifications

Exception Handling (25%)

- Comprehensive exception scenarios
 - Proper exception hierarchy
 - Graceful error handling and recovery
-

Constraints

- Use Java 8 or higher
 - In-memory data structures only (no database)
 - No REST interface required
 - Testing framework: JUnit 5 or TestNG
 - Time estimate: 4-5 hours
-

Sample Scenarios to Test

1. **Happy Path:** Submit matching buy and sell orders, verify trade execution
 2. **Validation Failures:** Orders from restricted countries, exceeding amount limits
 3. **FIFO Matching:** Multiple orders at same price, verify timestamp ordering
 4. **Partial Fills:** Large order matching with multiple smaller counter-orders
 5. **Concurrency:** 100 threads submitting orders simultaneously
 6. **No Match:** Orders with incompatible prices remaining in order book
-

Submission Guidelines

- Clean, compilable code
- All tests should pass
- Code should be well-commented where necessary
- Follow standard Java naming conventions
- Include a README with setup instructions